# Profile Generation and Guided Optimization for Compiled Queries

Project Proposal Document
April 1, 2021

Abi Kim, Kyle Dotterrer,
Wan Shen Lim
Carnegie Mellon University

## 1. INTRODUCTION

This document describes our proposal for our final project in CMU's 15-745: *Optimizing Compilers* course. The remainder of this document is structured as follows. First, in Section 1, we provide some group metdata. Next, in Section 2, we present our project proposal. Section 3 provides logistical information related to our group plans to conduct the project. We conclude in Section 4 with a brief literature review.

## 2. METADATA

The members of our group are:

- Abigale Kim (abigalek@cmu.edu)
- Kyle Dotterer (kdottere@cs.cmu.edu)
- Wan Shen Lim (wanshenl@cs.cmu.edu)

Project website: https://turingcompl33t.github.io/compilers-final/.

## 3. PROJECT DESCRIPTION

The following section provides a description of our project.

### 3.1 Project Goals

The goal of our project is to improve the runtime performance of query execution in a data-centric code generation DBMS engine by integrating dynamic optimization techniques. Specifically, we aim to implement profile-guided optimization for JIT-compiled queries in the NoisePage [2] LLVM-based execution engine. We identify the following subtasks:

- Implement support for collection of low-level statistics and profile information in the NoisePage JIT. These statistics may include runtime information for individual query pipelines, enabling the implementation of the SQL **EXPLAIN ANALYZE** command. More importantly, this profile information generated through this implementation will be used as input to other components of the system.
- Implement a custom LLVM pass (or passes) that incorporates profile information and the LLVM IR itself to more aggressively optimize hot paths.
- Implement a feedback loop mechanism that allows for an automated feedback-driven exploration of the space of potential LLVM optimization passes, based on the profile information generated by prior iterations.
- Explore techniques for efficient instrumentation of fused operators. Sufficient gains in efficiency might allow us to gather and utilize profile information in an online setting, rather than merely across queries.

We foresee challenges in this project arising from the following sources:

- **Operator Fusion** In a typical DBMS, **EXPLAIN ANALYZE** is simpler to implement because each operator is clearly demarcated. However, with the introduction of relaxed operator fusion in our execution engine, it is no longer clear what each operator is contributing to an overall pipelineâĂŹs work. Existing DBMS code-generation engines such as HyPer simply do not support **EXPLAIN ANALYZE**. The SingleStore engine implements something similar in their **PROFILE** command, but naturally this implementation is closed-source. How does one instrument pipelines of fused operators?
- **Optimization Pass(es)** We need to determine the appropriate pass or passes to implement that will be able to make effective use of the profile data we intend to gather. We assume that only a subset of available profile-guided optimization passes will be applicable in the context of our project.
- **Profile Data Collection** When profile-guided optimization is typically implemented in 'traditional' ahead-of-time compilers, profile information is written to an external file on the local filesystem. For our purposes, such an approach may be inappropriate because of the prohibitive runtime overhead of file IO. However, persisting the profile information in memory only introduces its own set of challenges, the most salient being the total volume of memory that may be consumed by the profile information if it is maintained indefinitely.
- **The DBMS-Centric Context** Some salient points of the DBMS setting are:
  - Highly multi-user: no single query should starve the entire system of resources.
  - Time to first response matters: it is not acceptable to spend too long optimizing instead of quickly responding.
  - Adaptive execution allows more time to be spent compiling anyway: fast OLTP queries are run through an interpreter, long OLAP queries are compiled in the background and swapped in when compilation is complete. This model lends itself well to multiple stages of compilation with different optimization levels, we are unaware

of existing projects that draw a link between adaptive execution and profile-guided optimization.

## 3.2 Preliminary Implementation Details

Profile information that we are considering collecting includes:

- Function entry probes: count the number of invocations of each function
- Basic block probes: count the number of entries to each basic block
- Edge probes: construct a graph of common transitions
- Value probes: construct a histogram of typical values at program points (e.g. branches)

Guided optimizations that we are considering using or implementing on the basis of this profile information include:

- Full and partial inlining
- Function layout
- Speed and size decision
- Basic block layout (i.e. block packing)
- Code separation
- Virtual call speculation
- Switch expansion
- Data separation
- Loop unrolling

## 3.3 Metrics

The primary metrics are compilation time and end-to-end runtime. Other miscellaneous statistics such as static or dynamic instruction counts may be collected, however, there is no direct correlation between instruction count and overall query performance.

## 3.4 Targets

- **75%** A custom LLVM pass taking (profile information, TPL) as input, operating with handcrafted manual optimization passes that should be applied to hot/cold pipelines.
- **100%** An automated feedback-driven exploration of the space of LLVM optimization passes for any given hot/cold pipeline.
- **125%** Assuming that profiling is slow, an exploration of efficient instrumentation techniques (especially in the context of fused operators) to allow for online profiling.

# 4. LOGISTICS

The following section describes preliminary project logistics.

## 4.1 Critical Path

Initially, it appears that the availability of profile information is the limiting factor for other features of our project. However, we foresee various methods by which we may begin implementation of multiple project components in parallel. The custom LLVM pass must be able to perform (possibly handcrafted) optimization passes based on profiling data. This does not necessarily require support for generation of profile information because that pass may operate on notional or âĂIJdummyâĂİ data and assume that real profile information will be integrated at some future point. Similarly, the low-level components of the feedback loop mechanism may be designed, implemented, and tested on notional data. However, full support for profile data generation will need to be implemented before an end-to-end solution for optimization feedback is possible.

The critical path for the project depends on designing the interfaces between individual components. Before beginning implementation, we require answers to the following questions:

- What is a suitable format for the profile data generated by the instrumented code?
- How do other components in the system access the profile information once it is stored?
- What is the cost model used to determine the relative efficiency of individual optimization passes?

The two components will be connected with a common JSON data interchange format for now.

## 4.2 Division of Work

Approximately, we plan to divide the work of the project as follows:

- Building the optimization passes
- Building profiling infrastructure

## 4.3 Schedule

The tentative schedule for project design and implementation follows below.

**Week of 04/02 to 04/08**

- Abi: Literature search/theoretical reasoning/testing to determine possible optimizations that we would want to include in our feedback loop.
- Kyle: Literature review of profile-guided optimization passes. Determine tentative list of profile features that may be required for pass implementation.
- Wan: Examine the state of profiling generated code in NoisePage. This was done in an ad-hoc manner for another unrelated research project, so there are some impediments that need to be resolved to collect statistics in a principled manner.

**Week of 04/09 to 04/15**

- Abi: First pass at coding the feedback loop. Implement logic determined by literature search. Test, and change predictions, and repeat. Coding the feedback loop requires determining hot and cold paths, then determining optimizations for the hot paths.
- Kyle: Also code the feedback loop. Make a final decision regarding the passes that will be implemented. Finalize the set of profile features required to implement these passes. Begin implementation for profile-guided passes.

- Wan: Support the collection of execution time at the granularity of pipelines. (i.e., entire fused operators). Attempt to support the collection of compilation time as well.

**Week of 04/16 to 04/22**

- Abi: Work on milestone report. Include more optimizations given granularity of instrumentation. Modify feedback loop as necessary given more information.
- Kyle: Work on milestone report. Continue work on profile-guided optimization passes. Begin integration with data collected from actual profiling.
- Wan: Work on milestone report. Support collecting both compilation and execution time at a pipeline granularity. Attempt to refine the granularity of instrumentation.

**Week of 04/23 to 04/29**

- Abi: Flexible. Continue to modify the feedback loop as necessary given more information.
- Kyle: Flexible
- Wan: Flexible.

**Week of 04/30 to 05/06**

- Abi: Work on final report. Flexible. (possibly collect more data/testing/results for final report).
- Kyle: Work on final report.
- Wan: Work on final report. Flexible.

## 4.4 Milestone

By 4/22, we aim to have achieved our 75% goal.

## 4.5 Required Resources

The project will be implemented in the NoisePage DBMS. This project is open-source software, implying that we already have the access we need.

We do not foresee any additional hardware requirements for this project.

At present, we believe that we have all of the resources necessary to conduct this study.

## 4.6 Getting Started

No work has been completed thus far on this project other than writing up this proposal. There are no impediments to beginning work on this project immediately.

# 5. LITERATURE REVIEW

While not the central focus of this project, the NoisePage execution engine plays a crucial role in our study - it is the substrate in which our project is implemented. Accordingly, knowledge of both the general concepts of query compilation as well as the details specific to NoisePage's implementation is an important prerequisite prior to beginning work on the project. Relevant background information regarding query compilation is available in [8], [4], and [7] among others. Descriptions specific to the NoisePage execution engine are provided by [9] and [6].

We conducted a brief survey of database systems that implement query compilation in a manner similar to NoisePage. We find that most state-of-the-art systems do not implement a feature analogous to **EXPLAIN ANALYZE**. The HyPer DBMS is an example of such a system [1]. This is likely a result of the aforementioned difficulty that accompanies the implementation of this feature in data-centric code generation engines. However, one system, the SingleStore DBMS does appear to support functionality similar to that provided by **EXPLAIN ANALYZE** in the form of its **PROFILE** command [3].

One important piece of information that we currently lack is a comprehensive review of techniques available for optimizations enabled by profile information. In this class, our primary focus has been on optimization passes that operate as pure functions of the input code. For this project, however, one of the contributions we hope to make is an optimization pass (or passes) that use profile information to apply optimizations that would otherwise be impossible. In the lecture on Dynamic Compilation, we reviewed two specific profile-guided optimization passes: partial dead code elimination and partial escape analysis [10, 5]. While these optimizations give us some idea of the style and structure of the optimizations that might be enabled by profile information, the degree of their applicability to our domain is questionable. We will need to conduct a more thorough survey of existing techniques in this space to decide upon the specific passes that are most relevant to our project.

# 6. REFERENCES


[1] HyPer. http://hyper-db.de/.

[2] NoisePage. https://noise.page.

[3] SingleStore. https://www.singlestore.com/.

[4] K. Krikellas, S. D. Viglas, and M. Cintra. Generating code for holistic query evaluation. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 613–624. IEEE, 2010.

[5] H. M. Lukas Stadler, Thomas Wurthinger. Partial escape analysis and scalar replacement for java.

[6] P. Menon, A. Ngom, L. Ma, T. C. Mowry, and A. Pavlo. Permutable compiled queries: Dynamically adapting compiled queries without recompiling, 2020.

[7] T. Neumann. Efficiently compiling efficient query plans for modern hardware. *PVLDB*, 4(9):539–550, 2011.

[8] D. Paroski. Code generation: The inner sanctum of database performance. http://highscalability.com/blog/2016/9/7/code-generation-the-inner-sanctum-of-database-performance.html, September 2016.

[9] A. P. Prashanth Menon, Todd C. Mowry. Relaxed operator fusion for in-memory databases: Making compilation, vectorization, and prefetching work together at last, 2017.

[10] J. Whaley. Partial method compilation using dynamic profile information, 2001.